Progetto Finale di Reti Logiche

a.a. 2021/2022

Daniele Asciutti
codice persona: 10678078
daniele.asciutti@mail.polimi.it

Indice

1 Introduzione

1.1 La specifica del problema

Lo scopo del progetto è la realizzazione di un componente hardware per svolgere la convoluzione di una sequenza di bit con tasso di trasmissione pari a $\frac{1}{2}$; quindi, data una sequenza di n bit, sarà fornita in uscita una sequenza di 2n bit.

Il componente sviluppato si interfaccia con una memoria RAM con indirizzi di 16 bit e celle da 8 bit, da cui legge prima il numero di byte che verrano dati in input e poi sequenzialmente tutti i valori.

Sempre sulla RAM, dopo ogni elaborazione, scrive i nuovi valori partendo dall'indirizzo 1000.

INDIRIZZO	VALORE	NOTE
0	15	`numero di byte`
1	4	`primo byte`
2	138	`secondo byte`
3	204	`...`
4	64	
5	64	
6	204	
7	89	
...	...	

Figure 1: Rappresentazione della memoria RAM

1.2 Un esempio di convoluzione

Si prende come esempio la serie di byte sottostante e si svolge il processo di convoluzione.

INDIRIZZO	VALORE	NOTE
0	2	`numero di byte`
1	162	`primo byte`
2	75	`secondo byte`

INDIRIZZO	VALORE	NOTE
1000	208	`primo byte in output`
1001	205	`secondo byte in output`
1002	247	`terzo byte in output`
1003	210	`quarto byte in output`

Figure 2: A sinistra i valori di input della macchina e a destra i valori di output corrispondenti.

Per calcolare il valore di output viene utilizzata la seguente macchina a stati, con stato di reset `S0`.

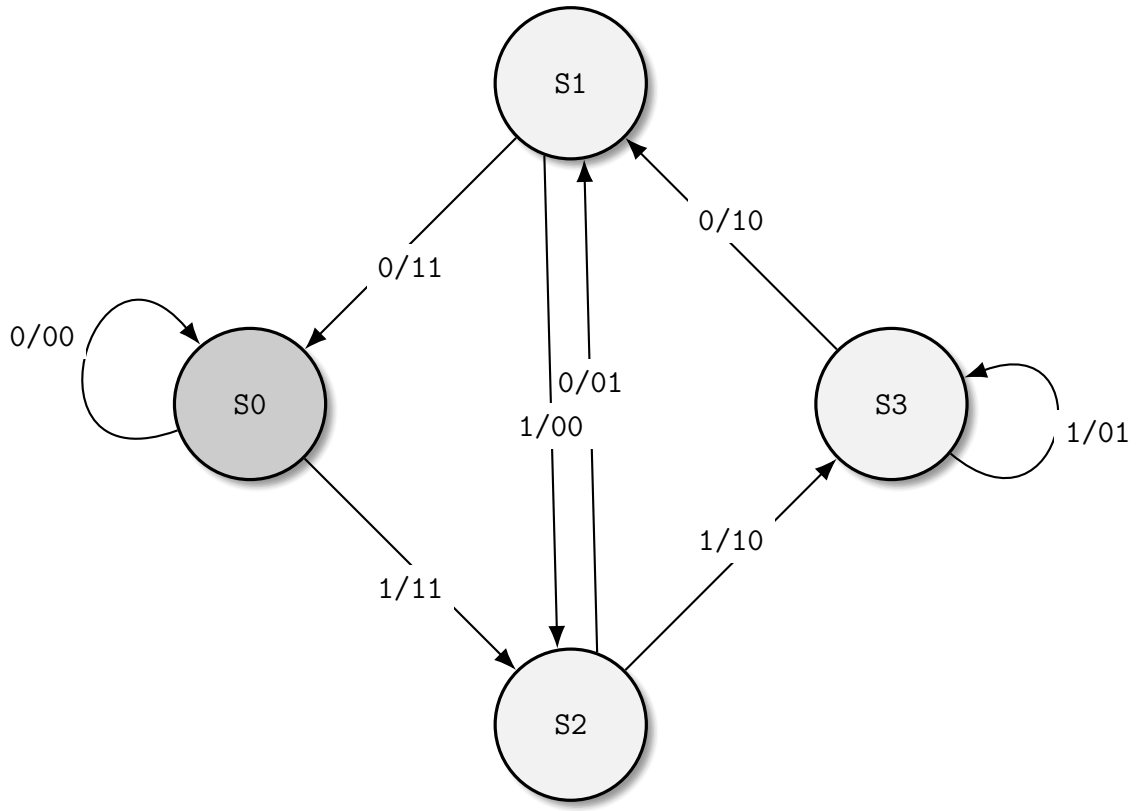


Figure 3: Rappresentazione della macchina a stati del convolutore.

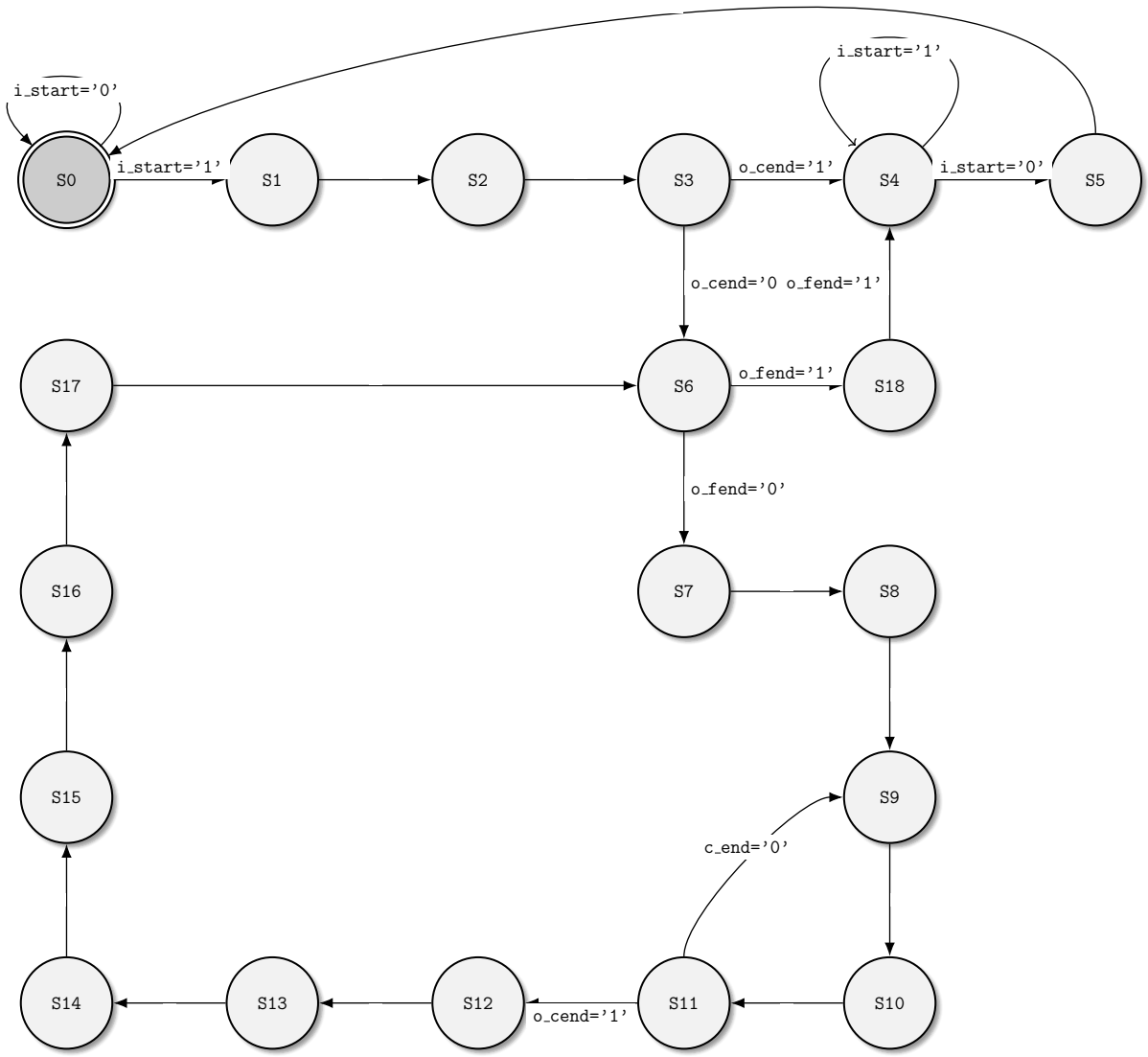
2 Architettura

Il processo di convoluzione inizia leggendo dall'indirizzo zero il numero di byte che verrano dati in ingresso, e poi per ogni byte si svolgono due fasi:

- inizialmente viene letto il byte in input;
- in un secondo momento, vengono calcolati i bit dal convolutore e si inizializzano i byte che poi verranno trasmessi in output.

Il comportamento del modulo implementato rispecchia quello di una Macchina a Stati Finiti (FSM).

2.1 La macchina a stati finiti



2.1.1 Prima fase

Partendo dallo stato di `S0`, in cui tutti i segnali corrispondono al loro valore di default, si attende che il segnale di start venga portato a "1". Nello stato `S1` si impostano i segnali per la lettura del primo byte che verrà poi letto nello stato successivo e, contemporaneamente, si inizializzano i due registri `reg2` e `reg3` che serviranno come contatori per il funzionamento della macchina, e che corrispondono rispettivamente all'indirizzo di input e output di memoria (`o_inmem` e `o_outmem`) . Dopo ciò, nello stato `S3` si verifica che il byte letto ha valore diverso da "0" e si procede con la seconda fase (stato `S6`); se il valore è uguale a "0", invece, si procede verso la conclusione (stato `S4`).

2.1.2 Seconda fase

All'inizio della seconda fase, nello stato `S7` si impostano i segnali per la lettura del byte all'indirizzo `o_inmem`, che viene letto nello stato `S8`. Negli stati successivi (stati `S9,S10,S11`), per ogni bit del byte viene comandato il convolutore (Figura 3) il cui output viene scritto in `reg6`. La macchina continua l'elaborazione e negli stati `S13` e `S16` manda in uscita all'indirizzo `o_outmem` i byte appena calcolati; gli stati `S12,S14,S15` servono a supporto di questa operazione, infatti hanno il compito di gestire il corretto valore di `o_outmem` per i due output. Infine, lo stato `S17` incrementa i contatori degli stati `reg2` e `reg3`, e lo stato `S6` controlla se l'elaborazione è conclusa (stato `S18`) o se ci sono ancora altri byte in input (stato `S7`).

2.1.3 Conclusione dell'elaborazione

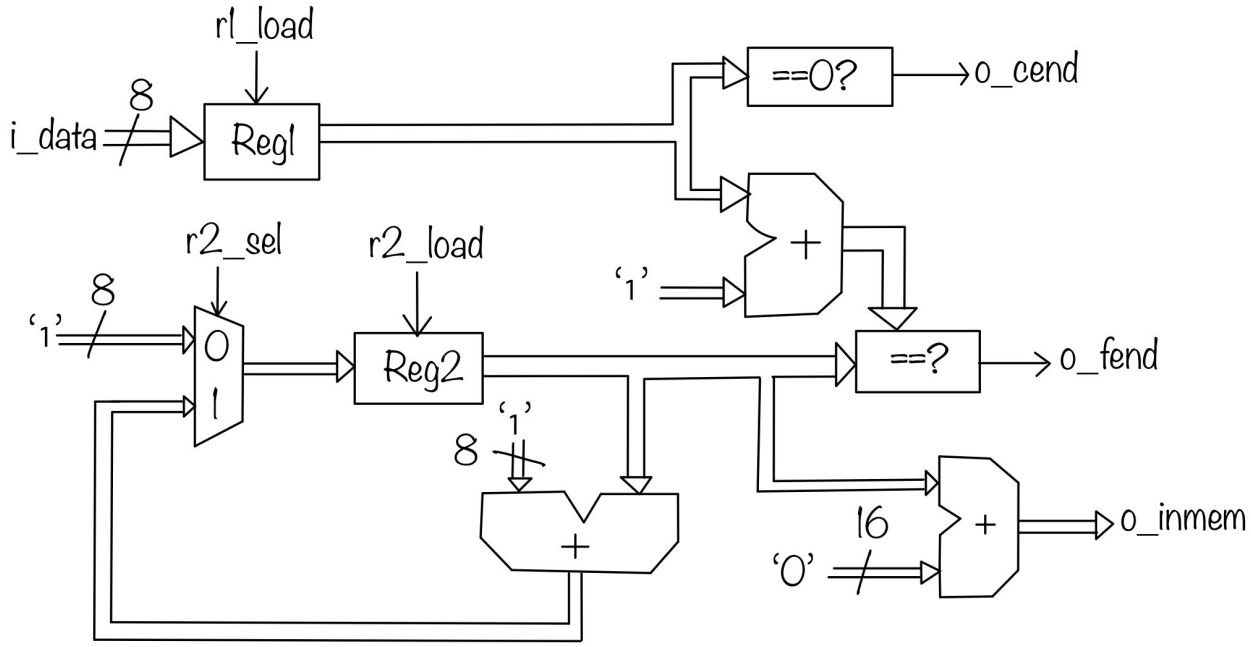
Quando la macchina a stati arriva allo stato `S18` vuole dire che l'operazione è conclusa; viene chiamato il comando di reset del convolutore e si passa allo stato `S4`. In questo stato si procede con la segnalazione di elaborazione finita (`o_done="1"`) e si attende che `i_start` sia riportato a "0". Nello stato `S5` viene riportato `o_done` a "0" e si passa allo stato `S0` da cui poi potrà ricominciare una nuova elaborazione.

2.2 DataPath

Per semplicità di lettura il Datapath è stato diviso in tre sezioni principali: input iniziale, contatore indirizzo di output, processo di convoluzione.

2.2.1 Input iniziale

Viene preso in input il primo byte della sequenza e si attiva un contatore che gestisce la durata dell'elaborazione e l'indirizzo di input da memoria.

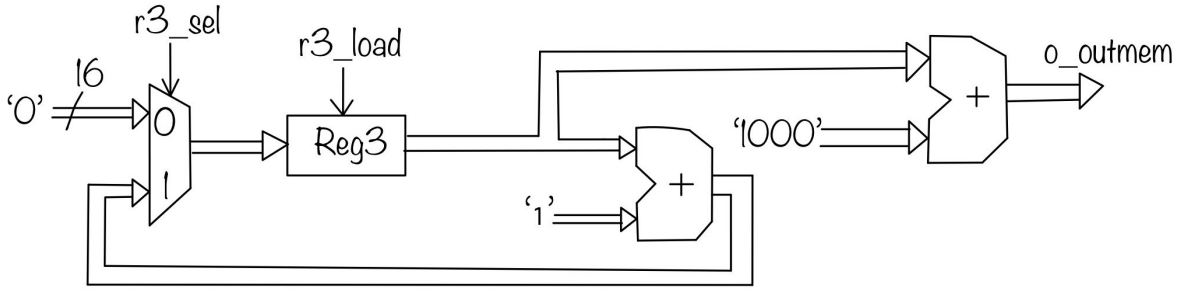


Reg1: Registro in cui viene caricato il primo byte di ogni sequenza.
r4_load: Se '1' copia il valore di i_data.
Reg2: Registro usato per mantenere il valore dell'indirizzo in input.
r2_load: Se '1' copia il valore uscente dal multiplexer.
r2_sel: Se '0' resetta il contatore, se '1' continua il conteggio.
o_cend: Se '1' la FSM va in fase di conclusione dell'elaborazione.
o_fend: Se '1' la FSM va in fase di conclusione dell'elaborazione.
o_inmem: Indirizzo di input da memoria.

2.2.2 Contatore indirizzo di output

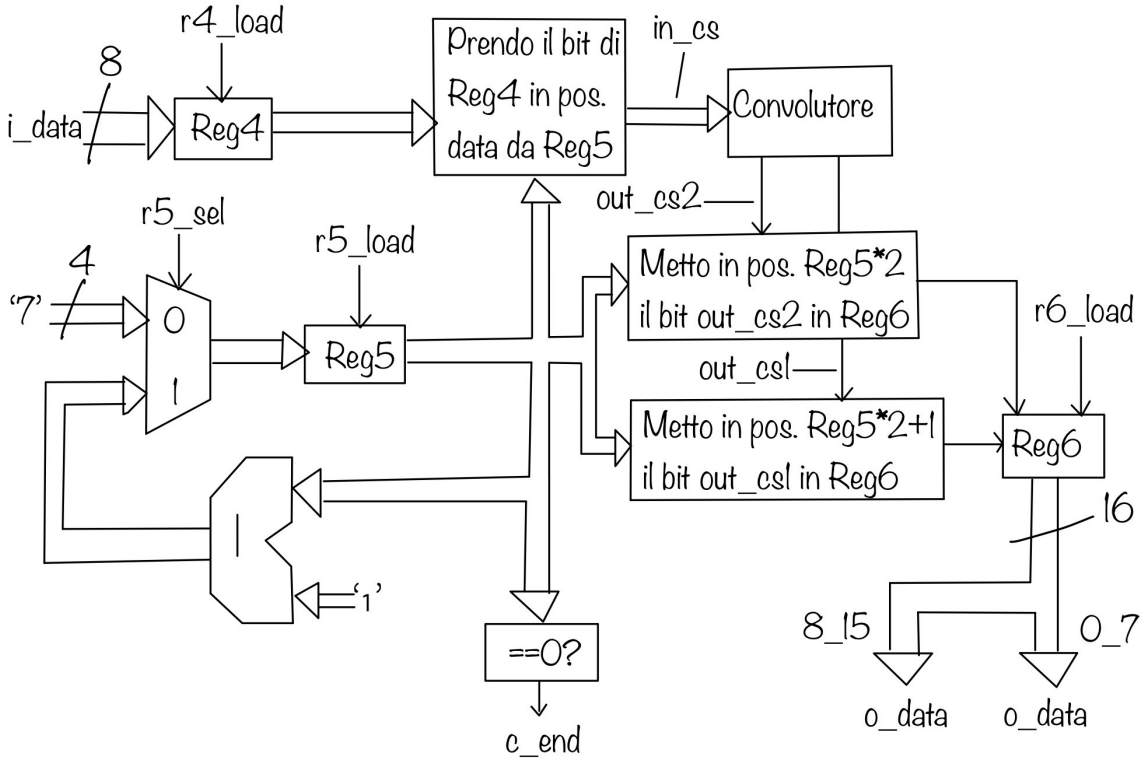
Contatore che ha lo scopo di gestire l'indirizzo di output in memoria.

Reg3: Registro per salvare il valore del contatore.
r3_load: Se '1' copia il valore uscente dal multiplexer.
r3_sel: Se '0' resetta il contatore, se '1' continua il conteggio.
o_outmem: Indirizzo di output da memoria.



2.2.3 Processo di convoluzione

Viene preso in input ogni byte della sequenza e, grazie al contatore, ogni bit è processato e poi inserito nel registro per formare i byte da caricare in memoria.



Reg4: Registro per salvare il valore del byte in input.
r4_load: Se '1' copia il valore del byte in input.
Reg5: Registro per salvare il valore del contatore.
r5_load: Se '1' copia il valore uscente dal multiplexer.
r5_sel: Se '0' resetta il contatore, se '1' continua il conteggio.
Reg6: Registro per salvare i byte di output.

r6_load: Se '1' carica i bit in ingresso.
in_cs: Singolo bit da dare in input al convolutore.
out_cs1: Primo bit in output dal convolutore.
out_cs2: Secondo bit in output dal convolutore.
c_end: Se '1' tutti i bit sono stati processati e si passa alla fase di output.

2.3 Implementazione in VHDL

La FSM è implementata con cinque processi:

- Il primo si occupa della gestione degli stati successivi dovuti al fronte di salita del clock, delle due fsm implementate e dei loro rispettivi reset;
- Il secondo imposta lo stato successivo della fsm implementata;
- Il terzo imposta lo stavo successivo del convolutore;
- Il quarto si occupa di tutta le gestione dei segnali nei rispettivi stati;
- Il quinto si occupa della gestione di tutti i registri presenti nella macchina.

3 Risultati Sperimentali

3.1 Sintesi

La sintesi si completa senza warnings o errori e, dall'*utilization report*, risulta quanto segue.

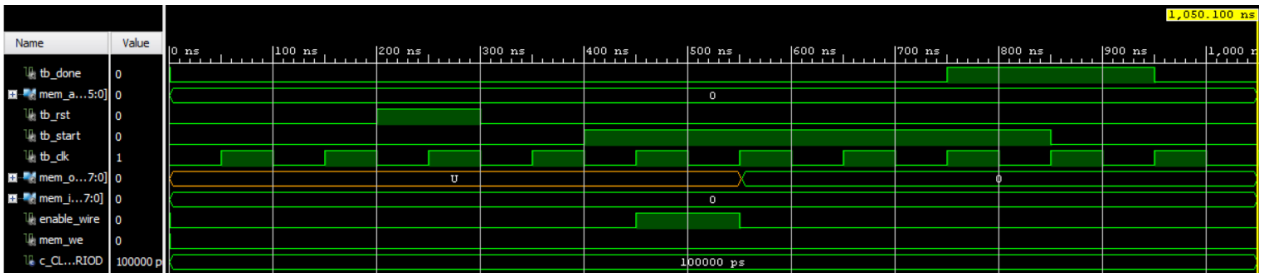
Site Type	Used	Available	Util %
Slice LUTs	282	134600	0.09
LUT as Logic	120	134600	0.09
LUT as Memory	0	46200	0.00
Slice Registers	67	269200	0.02
Register as Flip Flop	67	269200	0.02
Register as Latch	0	269200	0.00
F7 Muxes	3	67300	<0.01
F8 Muxes	0	33650	0.00

4 Simulazioni

Con lo scopo di verificarne l'effettivo funzionamento, la macchina è stata sottoposta a diversi test, molti dei quali casuali e alcuni scritti manualmente riguardanti i casi limite. Tra questi, sono stati selezionati e approfonditi cinque particolari situazioni: il caso in cui il byte all'indirizzo "0" è uguale a "0", quello in cui il byte all'indirizzo "0" è uguale a "255", il caso in cui si hanno più elaborazioni consecutive, il reset asincrono e, infine, reset multipli.

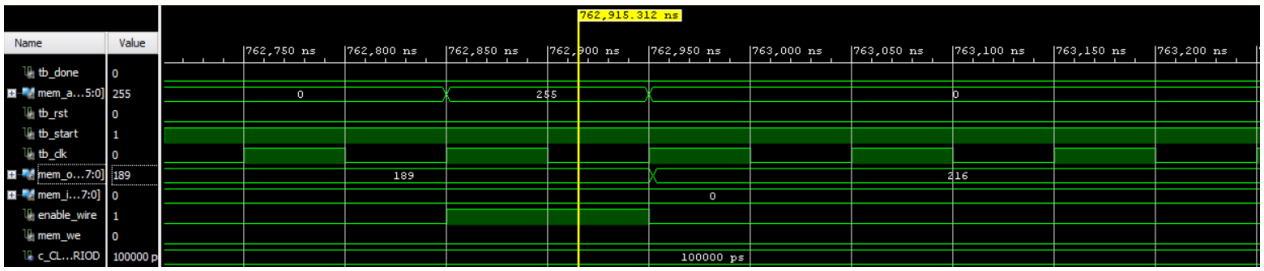
4.1 Sequenza minima

Nel caso in cui venga inserito il primo byte uguale a "0", il segnale `o_cend` viene impostato a '1' e quando la macchina arriva allo stato `S3` continua verso lo stato di conclusione `S4`.



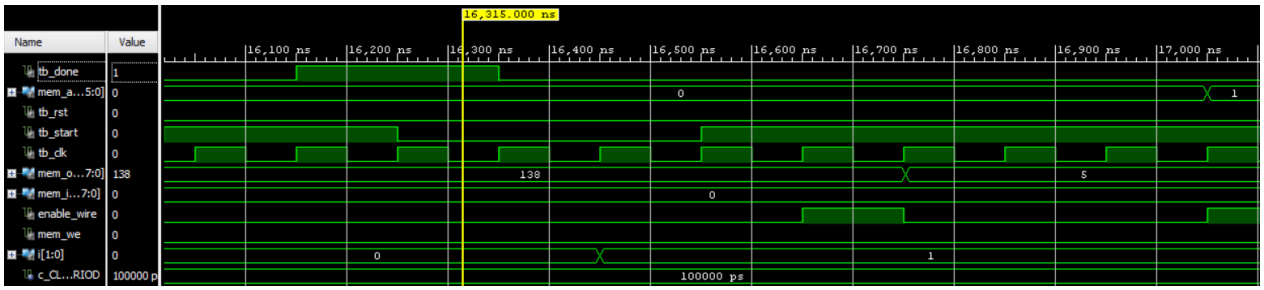
4.2 Sequenza massima

Se viene inserito il primo byte uguale a "255", la macchina si comporta normalmente procedendo a tutte le sue elaborazioni. Infatti la FSM è stata progettata con lo scopo di gestire numeri non maggiori di 255.



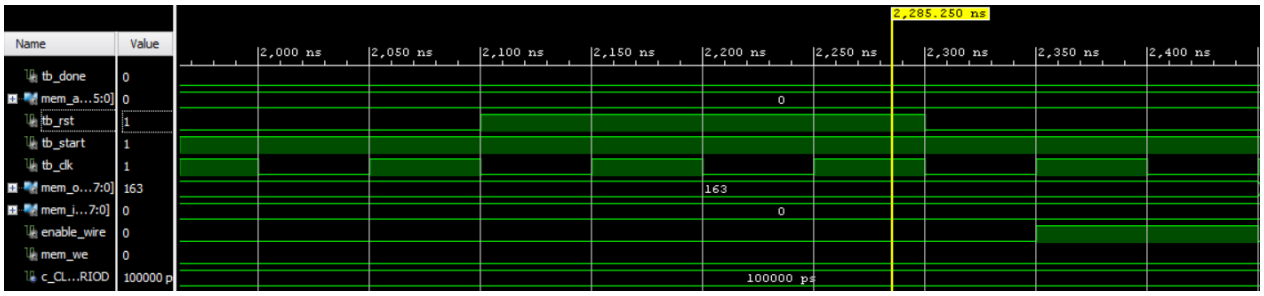
4.3 Più elaborazioni consecutive

Nella possibilità in cui vi siano più sequenze di byte in ingresso, sono stati creati dei test che verifichino il corretto funzionamento della FSM sia nel caso in cui avvenga un reset tra una sequenza e l'altra, sia nel caso in cui questo non si verifichi.



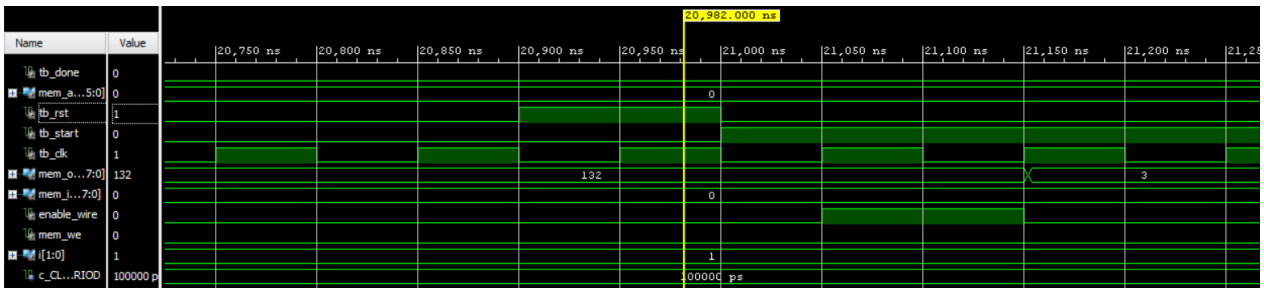
4.4 Reset asincrono

Vi 'e la possibilità che, durante il regolare funzionamento della FSM, il segnale di `i_rst` venga portato ad '1' con una tempistica non sincrona al segnale di clock. Allora, il processo si risveglia e la FSM torna allo stato di `S0`, indipendentemente da quello in cui si trova e dalle azioni che sta svolgendo.



4.5 Reset Multipli

Per estremizzare il test sul reset asincrono, sono stati testati anche casi in cui avvengono più reset in diverse sequenze consecutive; quando il reset avviene, viene riportata la FSM nello stato iniziale `S0` ed è quindi pronta a iniziare una nuova elaborazione.



5 Conclusioni

In conclusione, lo sviluppo di questo progetto ha portato all'acquisizione e al consolidamento di conoscenze base sulla progettazione e sulla sintesi di componenti logici. Le conoscenze teoriche assimilate durante il corso sono state quindi applicate in un contesto pratico.